Logical validation of safety and control system specifications against plant models

Taylor, J.R.

Publication date:
1981

Document Version
Publisher's PDF, also known as Version of record

RISØ-M-2292

LOGICAL VALIDATION OF SAFETY AND CONTROL SYSTEM SPECIFICATIONS AGAINST PLANT MODELS

J.R. Taylor

Abstract. Testing of computer based control systems is difficult, especially because of the problem of errors in specifications. By testing software in conjunction with a model of a plant, this problem can be overcome to a large extent. By extending this idea to an abstract domain, in which tests are performed symbolically, whole classes of errors can be discovered with each test. The paper describes the basic theory for such testing.

May 1981
Risø National Laboratory, DK-4000 Roskilde, Denmark

ISBN 87-550-0767-8
ISSN 0418-6435

Risø repro 1981

TABLE OF CONTENTS

INTRODUCTION

This paper presents a view of control system and safety system validation which is oriented towards problems of design and specification error. In particlar it treats the problem of design and specification errors in computer programs intended for safety systems. This is an area of pressing importance. Many kinds of computer based safety systems are already in use. Yet virtually all approaches to validation of software for nuclear and chemical plant safety systems, for transport and for medical systems, founder on the problem of specification errors.

Proof of program consistency with formal specifications has been promulgated as a way of preventing software errors [1]. But the derivation of formal specifications needed for such proof has itself proved very error prone [2]. Some 30-60% of software errors remaining in working control system software have been found to originate from specification errors [3].

There are several ways in which a design could be validated (fig. 1). a) The design as it is realised can be checked against the real world system to be controlled or against some kind of simulation model of the real world system. b) Specifications can be checked against a model of the real world system. c) The design can be checked against its specification. d) And if a model is to be used, it must be validated by comparison with a real world system.

The role of the plant model in validation of safety systems is considered to be fundamental. Systems <u>can</u> be specified functionally by describing their required input/output behaviour. But whether a system is acceptable depends on what it does in the plant to be controlled. The ultimate test is what it achieves in the real system in the face of design errors, implementation errors <u>and</u> specification errors. The importance of validation against <u>models</u> of the plant rather than the real plant rests on three grounds. First, the designers mental model

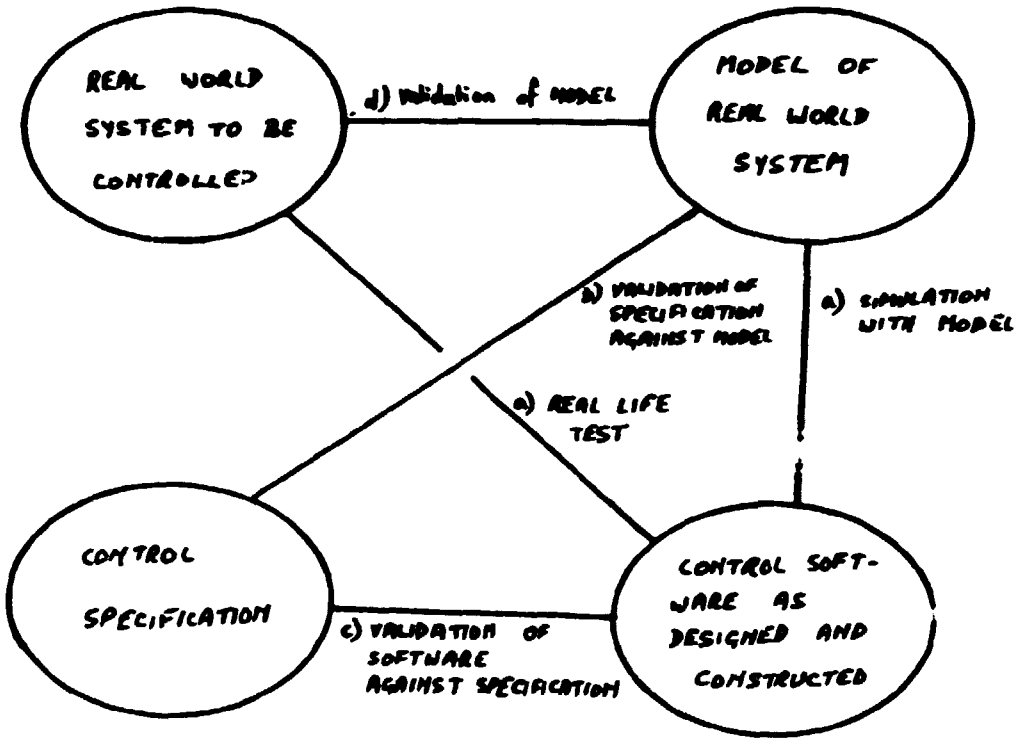


Fig. 1. Types of validation.

of the plant is a prime source of design information, and is available at an early stage in the design process. Second, the model is available to mathematical analysis, whereas the real plant is not. Third, the plant model provides a way of validating safety before a control system is applied to a real plant.

In validating safety and control systems, the plant model plays several roles.

- It allows specifications to be stated at a high level, in terms of the plant rather than the control system.

- Incorrect performance of the complete system (control+ safety+plant systems) can often be detected by inspection, whereas detection of errors in performance of the control system alone may be harder.

- It allows test cases to be derived systematically, so that these can later be applied to the real system.

- If the model is contructed by a person other than the control system designer, then this provides a very subtle form of redundancy, vulnerable to the absolute minimum of common causes of error.

It is not considered here that there is a fixed sequence of specification, design, implementation, plant modelling, and validation. Rather, it is known that increase in knowledge of the plant to be controlled, and development of specifications, proceeds in parallel with software implementation in most cases. Hence there is a need for final validation of plant models and specifications immediately prior to use or real system tests of control or safety software.

Testing or validation can proceed by considering specific cases of control system performance, and checking them against a specification, against a model, or against the real system. If this case by case approach is taken, then either some way is needed for selecting cases systematically, or a great many cases must be tested. Except for the simplest systems, this will generally require automatic testing if a reasonable level of reliability is to be demonstrated [4]. Alternatively, analytical forms of validation may be used [5]. As will be seen, these generally correspond to evaluation of sets of cases rather than individual cases of plant performance.

Specifications

The kinds of specifications described here will be of four types.

1. Functional specifications - descriptions of what a system should do.

2. Safety specifications - descriptions of what a system should not do.

3. Performance specifications - descriptions of how well a system should perform, in terms of a performance criterion.

4. Reliability or risk specifications - statements of the maximum probability with which specified unwanted events may occur.

(Other specification types are possible).

Functional specifications are particularly important for the methods described here. The language for these is fundamental to functional validation.

Essentially a functional specification states what a system should do. It can be expressed by statements such as:

- When A happens, X, Y, Z ---- should happen.
- When A happens, if P is true, X, Y, Z should happen.
- When A happens, X then Y then Z should happen.
- A should never happen.
- A should never happen while Y is true.
- P should always be true.
- Q should never be true.

In these statement types, the terms A, X, Y, Z happens may be events or actions describing specific changes of state, for example 'button 12 is pushed', or 'pressure becomes high', or 'tank 1 becomes filled'. Events described in this simple way imply a finite state model of the plant. Terms like P in these statements may be conditions such as 'pressure is high'.

As an alternative to these simple finite state descriptions of events and conditions, more flexible description forms may be used, for example:

- Continuous state discrete time models - here state variable values are given by real numbers, but changes are described as happening at specific points in time, e.g. 'when pressure becomes greater than 10 MPa, relief valve opens'.

- Continuous state and continuous time, behavioural, e.g. when A happens pressure follows trajectory T, --------

- Continuous state, continuous time, analytically described, e.g. 'when button 12 is pressed pressure should = kt^2'.

As can be seen from the examples, there is no particular problem in combining these specification types. There may though, be considerable difficulties in drawing analytical conclusions concerning more complex specification types. Note that similar statement types can be used to describe aspects of plant models and control systems, if the word 'should' is removed. For example, 'when pressure difference across valve is ΔP, flow through valve is $K\sqrt{\Delta P}$'. A syntax for such a model description language is given in appendix 1.

The language described so far does not allow timing statements to be made. Terms like 'After <time delay>, For <time interval>, before <event>, after <event>, until <event>' may be added to complete this aspect of the specifications.

Performance specifications generally take the form of statements about the rate at which events should take place, or of mathimatical functions of state variables which are to be maximised, or are to exceed certain values, e.g. production rate > 1 ton per hour.

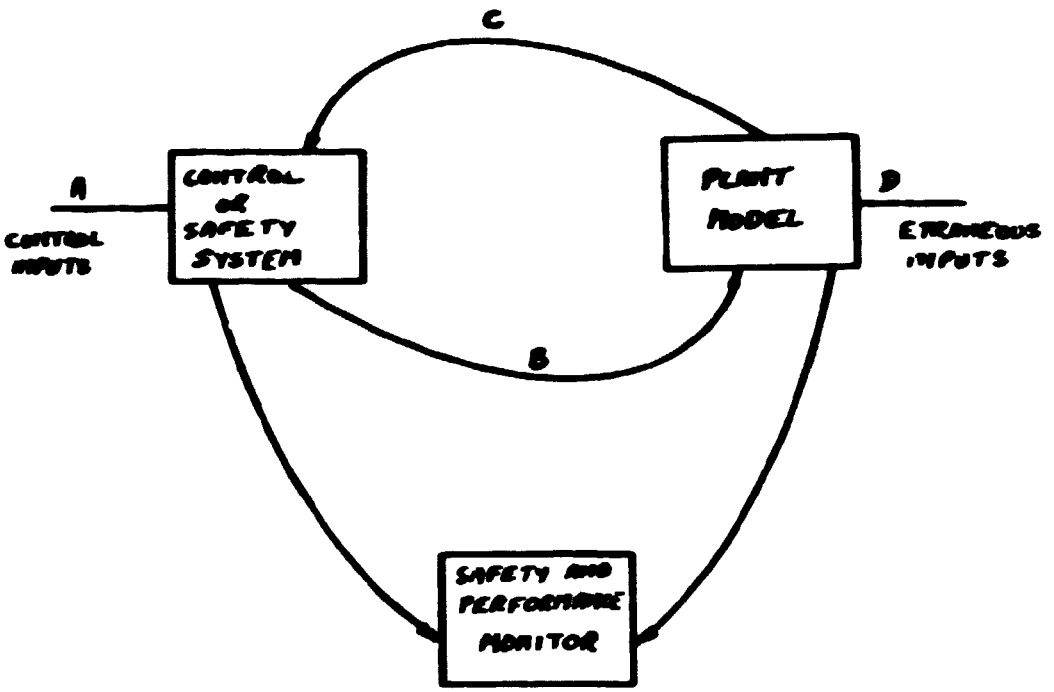
Whether the language proposed here is adequate for all purposes is not known. What is known is that it is adequate for very many practical applications.

Framework of testing a design against a specification and a model

The most straightforward way of checking a design against its specifications is by simulation. Inputs are provided to a simulator of the control system and the plant, or to the real control system and a simulator of the plant. The performance of the control system is monitored, to ensure performance is satisfactory.

Checking can be performed in an ad hoc fashion, just to see whether performance is acceptable to the designer. If very much simulation is to be performed though, it is practical to 'instrument' the simulation, that is, to convert the logical statements of safety and functional specifications to equivalent monitoring devices, and to apply these to the simulation (fig. 2). For example, a safety specification for a railway system is that two trains should never occupy one block section at one time. A detector for this condition is simple to implement and can 'ring a bell' if the safety criterion is breached in a simulation.

For complete validation of a system using simulation, it is necessary to provide all possible sequences of inputs under all possible initial starting states. This is theoretically feasible when finite state models are used in the simulation. It is not theoretically possible when continuous variable systems are used. In both cases there are very large problems in pro-



	SIMPLE RANDOM SIMULATION	SYMBOLIC SIMULATION
A	Random data selection	'systematic' input selection
B	Simulation result	results of symbolic execution
C	Simulation result	results of consequence analysis
D	Random data selection	'systematic' input selection

Fig. 2. General scheme for control system testing against a plant model

viding even an approximation to a complete simulation for practical systems. The number of inputs required is generally astronomical [4].

If a simulation approach is to be taken to control system validation it is necessary to be able to test whole classes of inputs at one time. The tests must be chosen so that by applying a single test, possibilities of errors or unacceptable performance for the entire class can be detected.

One way of doing this is suggested by the technique of 'symbolic execution' of computer programs [15]. Instead of working with real values for variables, the simulator should work with values described by symbolic formulae. The 'symbolic simulation' starts with a symbolic description of possible initial states of the plant, and investigates possible discrete changes of plant output/computer inputs. Applying the inputs to computer programs, it uses 'symbolic execution' to deduce computer outputs. Applying the outputs from the programs to the plant model it deduces the output from plant components resulting from application of symbolically described inputs.

To allow this symbolic simulation, the process of deducing outputs from inputs for plant components needs to be formalised. A state model of plant components allows this. Components models take the form of a set of transfer functions.

{input event + component state → output event + new component state}

Deductions in symbolic execution take the form

input event + component model + component state
→ output event + new component state.

This process has been formalised by the author as automated consequence analysis [6, 7, 8]. So far, only finite state versions of this analysis have been implemented.

Deducing an output event involves matching an input event to a transfer function, and extracting the output event from the transfer function. Matching a given formula describing an input event to one of a set of event transfer functions involves being able to match formulae to see whether they are identical, or whether one implies the other. If the given input event plus the given component formulae are identical to, or imply, the prefix of an event transfer function, the output event and new component state condition are deduced.

The simplest way of ensuring ease of matching between input events, component conditions and transfer functions, is to reduce all event formulae to a canonical form and to keep them in this form. The form which the author has found most useful is disjunctive normal form in which terms are grouped in sets ANDed together, and the ANDed groups are ORed together. The terms are sets of equalities or inequalities between state variables and finite state values.

The ability to derive canonical forms sets an effective limit to the freedom of language which can be used in specifications and model descriptions of this kind.

Canonical forms are also important in deriving a complete set of inputs for this kind of evaluation of system behaviour. The full range of values which the system input can take is described logically, and the description is reduced to a disjunctive normal form. Each of the OR'ed disjuncts can then be regarded as a 'test case' for symbolic simulation. As the simulation proceeds, it will be found that the further course of the simulation depends on plant state. As each alternative state (disjunct of the conditions describing the plant) is reached, a new sub case of the test is derived.

Monitoring of symbolic simulation for satisfaction of satefy specifications involves using the same kind of deduction techneques as for deducing plant component output from inputs. Such monitoring involves the difficulty that some deductions may be made to the effect that specifications may or may not be satisfied, without being able to say either way. For example, it

may be deduced that plant pressure becomes greater than 10 atm. A safety specification may be that it never becomes greater than 20 atm. It is not possible to decide whether the specification is satisfied without improving the model.

An alternative to symbolic simulation which works forwards in time, is to work backwards, making deductions from effect to cause. A statement concerning system state is chosen at some place in the system, and deductions are made backwards through the system. The deductions are made continuously until only external inputs to the system and initial states remain in the deductions. The final statement then represents the cause of the initial statement. By keeping different disjuncts separate, each disjunct represents a different class of inputs, a different input case. If the language given in appendix 1 is used, then each class is described by a set of equalities and inequalities.

This method has been implemented by the author and his colleagues using finite state models for process plant, as automatic fault tree construction [8, 9] and using weakest precondition theory for making deductions about software [10, 11].

If a neutral initial statement is made the starting point of this process, then the method serves to find the different classes of input resulting in fundamentally different modes of system behaviour. Each class is described by a set of polynomial inequalities. By solving the inequalities individual test valves can be chosen which guarantee a coverage of each fundamentally different type of behaviour of the plant model. By choosing many test points in each class, the probability that the system behaviour is described by an incorrect polynomial can be reduced to a very low value [12].

Alternatively, if the starting point for the backward deduction is made to be a (negated) safety specification or a functional specification statement, the conditions under which these specifications are not fulfilled can be found.

If timing is an important aspect of performance then symbolic simulations and backward deductions of causes must take into account the time at which various events occur, and the time at which various conditions are true. This corresponds tc setting up a set of 'histories' of plant state descriptions.

Checking specifications

The methods described in the previous section allow some aspects of specifications to be checked formally. Firstly, the self consistency of specifications can be checked, by reducing the overall set of specifications to canonical form.

Specifications	canonical reduction →	False	- inconsistency
		TRUE	- tautology
		Otherwise	- reduced spec. in canonical form

Secondly, the specifications can be checked against the model to see if there is any possibility of fulfilling them.

Safety specifications + model	backward symbolic simulation (Fault tree analysis) →	False	- error in model or model failure analysis produced
		TRUE	- specification always satisfied
		Otherwise	- further requirement allowing specification to be met.

Thirdly, the specifications can be checked with both plant model and control system as in the previous action, to yield cases where the specification is not met.

Specification + plant model + control system	→	False	- error in model or plant failure analysis or error in control system design
		TRUE	- specification always satisfied
		OTHERWISE	- further modelling required.

Some completeness criteria can also be established for specifications, for example that all outputs of a system should have their range of values specified. Unfortunately, it is not possible to provide general rules to guarantee completeness of sets of specifications, because this is a creative task. What can be done is to ensure 'closure' of the specification set, that is, that any particular specification type is applied uniformly. For example, if a state variable is specified for some point in time it should perhaps be specified at all times.

There are some general safety principles which can be applied at all times to any system, for example fail safe principles, and the principle that it should always be possible to bring a system back to its rest state. Making a collection of these principles would greatly strengthen the theory presented here.

Adequacy of models

If plant models are to play such an important role in validation of systems designs, then it is necessary to know how these will be developed and verified.

A fundamental principle should be that the model should be developed using a well developed theory of modelling, according to a standard procedure. If this is done, then the resulting models can be justified by reference to the general success of the standard procedure. Typical standard modelling procedures have been described elsewhere [13].

There will, even with the best modelling procedure, be many choices left open to the engineer. In particular the degree of detail in models must be chosen. An acceptable degree of detail should preferably be chosen by reference to earlier systems and accident reports. A general rule is that the plant model should be implemented at a level of detail sufficient to repredict system perfo: ance and accident types which have already happened.

Once the plant is built, the model needs to be validated before a full set of system tests begins, or before system validation is terminated. For these, system identification methods, or parameter adjustment optimisation methods are generally necessary [14] . The procedures are well known from development of analogue and hybrid computer plant models.

Remaining problems

The theory described here provides several routes to validation of control and safety system designs against a plant model and a set of system specifications. The result should be an internal validation of a system which is much more thorough than those in general use today. It is worthwhile enquiring, though, about the remaining possibilities for error.

One possibility is that models are implemented wrongly, or they are conceived incorrectly. There is always a possibility that the designer misunderstands the way in which his plant works. If the model is made independently of the control system then there is a good chance that errors will be detected. But there remains the possibility that the control system designer and the model constructor have similar misconceptions about the plant.

Another possibility for error is that the model is not made sufficiently detailed in all its aspects so that some parts of the specification are not checked. This is particularly important if some parts of the specification are implicit. Again it is important to be able to build up a library of different types of formal specifications so that this problem gradually can be covered by reference to standard cases.

One particular form of error in models and specifications is that of omitted special cases. For example, in specifying a tax system, it may be overlooked that the regent is generally treated as a special case. This type of error is not likely to be trapped by the techniques described here. But by allowing a very high level of language for specification it may be hoped that this kind of error may be more easily avoided.

REFERENCES

[1] C.A.R. Hoare. An asciomatic basis for computer programming. Comm. ACM 12, 10,1969.

[2] J. Goodenough. Towards a theory of test data selection. IEEE Trans. Software Engineering, Vol. SE1 No. 2, 1975.

[3] S. Wright. Private communication.

[4] W. Ehrenberger et al. Probability considerations concerning the testing of correct performance of a computer. IFAC Congress, Boston 1975.

[5] S. Bologna. Deriving test cases from specifications. Report RT/ING (78)3 CNEN.

[6] J.R. Taylor. A formalisation of failure mode analysis. RISØ-M-1654, 1973.

[7] J.R. Taylor. Sequential effects in failure mode analysis. RISØ-M-1740, 1974.

[8] J.R. Taylor and E. Hollo. Experience with algorithms for automatic fault tree and cause consequence analysis. International Conf. on Nuclear Systems Reliability, Gatlinburg, Tennessee 1977. (SIAM 1978).

[9] J.R. Taylor. An algorithm for fault tree construction. Notat N-19-80.

[10] J.R. Taylor and S. Bologna. SSPTV a software system for program test and verification.

[11] S. Bologna and J.R. Taylor. Validation of safety related software. IAEA Specialist Meeting on Computerised Safety Systems, Pittsburgh 1977.

[12] J.R. Taylor. A background to risk analysis. Available from The Library, Risø National Laboratory, DK-4000 Roskilde, Denmark.

[13] M. Lind. The use of flow models for automated plant diagnosis, to be published in: Human Detection and Diagnosis of System Failures, Jens Rasmussen and W.B. Rouse (Eds.), New York, Plenum.

[14] T.W. Kerlin, E.M. Katz, A.T. Chen, J.G. Thakkar, S.I. Chang. Dynamic testing in nuclear power plants for model validation. Transactions ASME, September 1976, p. 340-347.

APPENDIX. A LANGUAGE FOR PROCESS PLANT SPECIFICATIONS

Syntax for functional specification language.

```
<conjunct>          ::=  <variable> <condition operator>
                             <variable value>

<condition
 operator>          ::=  = | > | < | IS

<condition>         ::=  <disjunct> <OR <disjunct>>_o^n

<disjunct>          ::=  <conjunct> <AND <conjunct>>_o^n

<simple event>      ::=  <variable> <event operator> <variable value>

<event operator>    ::=  BECOMES |
                         BECOMES EQUAL TO
                         BECOMES GREATER THAN
                         BECOMES LESS THAN
```

Note: The language becomes much more powerful if <variable value> is replaced by <arithmetic expression>

```
<event>             ::=  <simple event> <AND <simple event>>_o^n

<event
 specification>     ::=   WHENEVER <event> IF <condition>
                                       THEN <event sequence>

<event sequence>    ::=  <delayed event><THEN <delayed event>>_o^n

<delayed event>     ::=  <AFTER <arithmetic expression>>_o^1 <event>>|
                         <event>

<condition
 specification>     ::=   WHILEVER <condition>
                          THEN      <condition> |
                                    <condition> ALWAYS
```

<simple action> ::= <subject><verb>
<subject><verb><object>
<subject><verb><<preposition><substantive>>$_{o}^{n}$

<subject> ::= <substantive>

<object> ..= <substantive>

<substantive> ::= name of concept defined in the model.

<action specification> ::=
<WHEN <event>>$_{o}^{1}$ <IF <condition>>$_{o}^{1}$
THEN <action sequence>

<action sequence> ::= <delayed action> <THEN <delayed action>>$_{o}^{n}$

<delayed action> ::= <AFTER <arithmetic expression>>$_{o}^{1}$
<extended action>

<extended action> ::= <action> <extent>$_{o}^{1}$

<extent> ::= FOR <arithmetic expression> |
UNTIL <event>

<verb> ::= OPENS, CLOSES, STARTS, FILLS, MOVES, --

<preposition> ::= FROM, TO, -----

Note: While the semantics of event and condition operators can be standardised, semantic interpretation of verbs and prepositions must be supplied.

For simple verbs such semantic descriptions can generally be given in terms of events, conditions and processes.

A process is a continuous change taking place over a period of time (a time which is long compared with time delays for 'events'.

Processes can be described in terms of a single verb, CHANGES.

```
<simple action> ::=  <variable> CHANGES  FROM <value>
                                          TO <value> |

                     <variable> CHANGES AT RATE
                                     (arithmetic expression)
```

For systems with internal structure, statements to describe this structure may be useful

```
<decomposition
 specification> ::=  <subsystem name> CONTAINS
                                 <part list>

<part list>     ::=  <subsystem name> <, <part list>>[1 o]

<connection
 specification> ::=  <subsystem name> IS CONNECTED TO
                                 <subsystem name> FROM <part name>
                                                   TO <part name>

<variable>      ::=  <variable name> OF <object name> |
                     <object name> <variable name> |
                     <object name> <part name> <variable name>
```

<subsystem specification>::=

<subsystem name> HAS PORTS
<port name> <, <port name>>$_{o}^{n}$ |

<subsystem name> HAS VARIABLES
<variable name> <, <variable name>>$_{o}^{n}$

<port specification> ::=

<port name> HAS VARIABLES
<variable name> <, <variable name>>$_{o}^{n}$

Risø National Laboratory

Risø-M-2292

Risø-M-2292

Title and author(s)

LOGICAL VALIDATION OF SAFETY AND CONTROL SYSTEM SPECIFICATIONS AGAINST PLANT MODELS

J.R. Taylor

Date: May 1981

Department or group: Electronics

Group's own registration number(s): R-4-81

23 pages + tables + illustrations

Abstract

Testing of computer based control systems is difficult, especially because of the problem of errors in specifications. By testing software in conjunction with a model of a plant, this problem can be overcome to a large extent. By extending this idea to an anstract domain, in which tests are performed symbolically , whole classes of errors can be discovered with each test. The paper describes the basic theory for such testing.

Copies to

Available on request from Risø Library, Risø National Laboratory (Risø Bibliotek), Forsøgsanlæg Risø), DK-4000 Roskilde, Denmark
Telephone: (03) 37 12 12, ext. 2262. Telex: 43116